Tcl warts

Tcl warts presents aspects of Tcl that a considered frustrating, confusing, aggravating, enraging, etc.

Quoting-related warts

Mandatory "quoting" of strings in expr

The following is not currently allowed:

```
set a hello
expr {$a eq hello}
```

There's no good reason. It's just a wart.

AMG: I'm inclined to disagree. There is good reason. The [expr] syntax is significantly different than the Tcl syntax, and in the [expr] language, strings must be quoted (using braces or double quotes, not backslashes on each character). That's how it's defined. This is done to avoid misinterpreting strings as operators or function names or other special characters such as whitespace and brackets.

Perhaps you're suggesting to treat unrecognized words as if they were quoted strings. Sure, this can be done, but it opens the door to surprises when a previously-unrecognized word becomes recognized, for instance when a new math function is added.

When it comes to language design, my preference is to be as picky as possible because it leaves room for future expansion. Every construct that is currently marked illegal, really is marked as reserved for future use. This is how we were able to get {*}.

PYK 2014-06-11: The "previously-unrecognized word" scenario could be avoided by having expr interpret anything that looks like a function call as a function call, and failing if it can't be found, but yes, I guess in order to know whether this is really a wart would require learning the rationale behind the decision to require double quotes or braces around string literals.

AMG: [expr] already has this behavior. Everything that looks like a function call (i.e. is an unquoted string followed by optional whitespace then open parenthesis then zero or more comma-delimited arguments then close parenthesis), it tries to invoke as a function call (i.e. calls [tcl::mathfunc::*name* {*}$*arglist*]), then fails if it can't be found.

glob

The need to somehow quote the glob {} combination so that one gets the quotes passed on to glob. Use quotation marks (") or braces around the arguments to get them past the parser.

Regular expressions

Like the problem with glob, the need to quote arguments appropriately so that various regular expression metas make it through the Tcl parsing.

exec Special Characters

Easily Tcl's most undeniable blemish: exec's brokenness in not directly allowing < or > as leading characters in arguments. open | has the same problem. See exec ampersand problem for discussion of a very similar problem involving &. Also, | doesn't work as the first character, and neither can the two-character sequence 2> be used.

{*} addresses another blemish, which was previously handled using eval and precise list manipulation.

LV: Shoot, exec doesn't allow | as a leading character in an argument either. And - as the leading character in an argument can cause problems, and so on...

On the other hand, if you are talking about using < or > as the leading character **in a file name**, you solve that problem in the same way you solve the others - make certain it is a relative path name:

```
% exec ls ./>stuff
./>stuff: No such file or directory
```

So it is doable.

AMG: - as leading character can cause two problems. If it's the leading character of the first character of the pipeline, it needs to be preceded by --, or else exec will think it's an option. If it's the leading character of other words, it may still require -- but for a different reason. It won't be exec misinterpreting the -, but rather the exec'ed command. Of course, each command is different, so -- isn't always appropriate.

"Quoting" (disabling special interpretation of) filenames with ./ works fine, but there's no way to do it for non-filenames, where the external command absolutely must receive <, >, 2>, |, or & as the first character of an argument. Since sh internally handles these characters, it also supports quoting them. Tcl doesn't handle these characters internally, so it falls to exec to do quoting, but exec doesn't. Adding quoting to exec would be highly unfortunate, considering how it would have to be used in practice. If you must start an argument with these special characters, construct your command pipeline as a suitably-quoted argument to "sh -c".

```
exec sh -c {echo "<urgent!!>"}
```

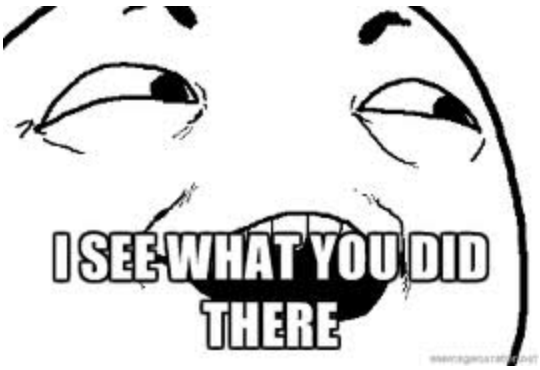
But watch out that the !'s don't get interpreted as history substitution markers!

Hmm, I bet there's a security issue here. If arbitrary untrusted user-provided data gets passed as the argument to an external program, what's to stop the user from passing >/etc/passwd and hosing the system?

Perhaps what's needed is an alternative to exec that doesn't use in-band signaling metacharacter sequences to configure command pipelines. For example, alternate between commands, each of which would be a list-formatted argument, and options describing how the commands are tied together. This can get complicated, so maybe justify the complexity of the syntax by also making it more powerful. Consider allowing for more flexible pipelines that branch different file descriptors around into a tree structure. E.g. some program is run, its stdout gets piped to the stdin of tee log, its stderr gets piped to the stdin of tee errors, and the stdout of each tee gets combined again to be the stdout of the whole pipeline. Or perform tee-like functionality internally, so that the stdout/stderr can be separately piped through gzip then written to disk but also combined to be the pipeline stdout. Lots of craziness is possible.

PYK 2014-05-29: 4 years after AMG's comments above comes TIP [L1], which fits AMG's description to a "tee".

AMG:



AMG's response to the quote warts

AMG: Most of these quoting-related warts arise from conflicts between the commands' little languages and the core language of Tcl defined in the dodekalogue. The same is true for sh, e.g. you need to quote wildcards in the -name option to the find program, or else the shell will expand them prematurely. Tcl has fewer metacharacters than sh, hence quoting isn't needed as much.

An example of a non-collision is format's % specifiers. They're similar to $ in that they substitute in the value of an argument (which is like a read-only variable), but % and $ don't collide simply because the notation uses a different symbol. An example of a collision is regexp's \ character, which works very much like, but not identically to, Tcl's \ character. The two collide because the symbol is the same. regexp could change \ to ` (which looks like the

upper half of a backslash), but this breaks compatibility and makes Tcl look more like Perl in that too many different typographical symbols are used. Also this isolated change would gain nothing without also changing regexp's [,], and $`. All this, just to avoid having to quote! And quoting would still be necessary for patterns containing quote or whitespace characters. Back to the format command: it's acceptable here to introduce a different symbol, since the notation is inherited from C.

Comments

Why can I not place unmatched braces in Tcl comments

namespace

See **An Anonymous Critique** [L2] in namespace

rwm: It seems that there are problems with procs named ":". especially inside namespaces... I was following the data-as-code example in implementing a forth parser with the command ":". searching the namespace for this command doesn't work as expected:

```
namespace eval z {proc : {} {puts "called ':' inside ::z"}
::z:
invalid command ::z:
::z:::
too many nested evaluations (infinite loop?
namespace eval z :
called ':' inside ::z
info procs ::z*
info procs ::z::*
::z:::
info procs ::z:::
namespace eval z {info procs :}
:
```

Hashes vs Arrays

Sometimes people attempt to simulate 2 or more dimensions of arrays using the Tcl associated hashes (aka tcl arrays). The gotcha here is that because the array index is a string, white space is significant.

```
$ set a1(1,2) abc
abc
$ puts $a1( 1,2)
can't read "a1( 1,2)": no such element in array
while evaluating {puts $a1( 1,2)}
```

Another gotcha here is trying to set arrays with white space:

```
$ set a1( 1,2) abc
wrong # args: should be "set varName ?newValue?"
while evaluating {set a1( 1,2) abc}
```

You need to use quotes if you are putting space into that variable.

AMG: Use nested dicts. Or, generate the array index using list $row $col instead of $row,$col. In all cases, be aware that $row and $col are general strings, not numbers.

Inconsistencies in names in Library

PYK is inclined to delete this one as there is no example.

AMG: This might be a consequence of Tcl 8.0's addition of the "object" API which uses Tcl_Obj instead of character pointers. Many functions had Obj added to the name, but it's not totally consistent. One example that comes to mind is Tcl_GetString() versus Tcl_GetStringFromObj() [L3].

KPV How about the inconsistencies with capitalization of package names? We have *package require Tcl* but *package require tdom*, *package require Tk* but *package require tile*.

Backslash-newline-whitespace replacement in brace-quoted words

AMG: I argue that this should not happen because brace-quoted words ought to be for verbatim text. See [L4] for details.

PYK 2017-07-01: Agreed. Among other things, this behaviour makes it painful to embed C in Tcl when there are multi-line function-like defines infolved.

Inconsistent error messages

AMG: Witness:

```
% puts
wrong # args: should be "puts ?-nonewline? ?channelId? string"
% binary asdf
unknown or ambiguous subcommand "asdf": must be decode, encode, format, or scan
% fileevent asdf fdsa
bad event name "fdsa": must be readable or writable
```

Sometimes we get "should be", sometimes "must be". Sometimes it's "unknown or ambiguous", sometimes just "bad".

I'm sure there are many more examples of inconsistency, but these few serve to illustrate the issue.

Bug reported here: [L5].

Similar bug reported here, now fixed: [L6].

APN Why is this worthy of being reported as a bug? I know of no system or application where this level of consistency is displayed, not any need for it (unlike in a RFC or specification). The error message should be clear to humans but that's about it. It would be different if the message was intended for program consumption but that role belongs to errorCode which I would agree should (or is it must?) have a consistent structure.

AMG: It came to mind because samoc was needing to parse some error messages due to lack of proper error codes. Yeah, I also reported the missing error code, but by then it was too late, I had seen the inconsistent error messages, and DKF did not push back against the reports. Yet.

encoding convertfrom Fails Silently

If the string is not valid in the selected coding, encoding convertfrom proceeds to munge the string and return it.

WITHDRAWN: [error] versus [return -code error]

AMG: Using [return -code error] gives a cleaner stack trace than [error]:

```
% proc err1 {} {error asdf}
% proc err2 {} {return -code error asdf}
% err1; set errorInfo
asdf
    while executing
"error asdf"
    (procedure "err1" line 1)
    invoked from within
"err1"
% err2; set errorInfo
asdf
    while executing
"err2"
```

What reason is there for [error] to inject itself into the stack trace?

AMG: Oh yeah, there is a reason. Having [error] in the stack trace shows on which line it was called. Never mind, this is totally valid and worthwhile.

DKF: I use **return -code error** for “you called this procedure wrong” and **error** for “I did something wrong, not you”.

See Also

Dangerous Constructs

gotcha

Updated 2020-04-01 17:52:25